Partial Synchrony Based on Set Timeliness

Marcos Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, Sam Toueg

Partial Synchrony Based on Set Timeliness

Marcos K. Aguilera
Microsoft Research Silicon Valley

Carole Delporte-Gallet
Université Paris 7

Hugues Fauconnier
Université Paris 7

Sam Toueg
University of Toronto

August 2, 2009

Abstract

We introduce a new model of partial synchrony for read-write shared memory systems. This model is based on the notion of *set timeliness*—a natural and straightforward generalization of the seminal concept of *timeliness* in the partially synchrony model of Dwork, Lynch and Stockmeyer [11].

Despite its simplicity, the concept of set timeliness is powerful enough to define a family of partially synchronous systems that closely match individual instances of the t-resilient k-set agreement problem among n processes, henceforth denoted (t, k, n)-agreement. In particular, we use it to give a partially synchronous system that is is synchronous enough for solving (t, k, n)-agreement, but not enough for solving two incrementally stronger problems, namely, $(t + 1, k, n)$-agreement, which has a slightly stronger resiliency requirement, and $(t, k - 1, n)$-agreement, which has a slightly stronger agreement requirement. This is the first partially synchronous system that separates between these sub-consensus problems.

The above results show that set timeliness can be used to study and compare the partial synchrony requirements of problems that are strictly weaker than consensus.

1 Introduction

The concept of partial synchrony, introduced in the seminal work of Dwork, Lynch and Stockmeyer [11], is based on the notion of *timeliness*, e.g., an upper bound Φ on relative process speeds: "in any contiguous interval containing Φ real-time steps, every correct process must take at least one step. This implies no correct process can run more than Φ times slower than another." In the partially synchronous systems in [11], *all* the processes are (eventually) timely relative to each other.

To define partially synchronous systems that are weaker than those in [11], but are still strong enough to solve consensus, the above notion of timeliness was later refined by considering the timeliness of each pair of processes *individually*. In particular, for shared memory systems, one can define the concept of *process timeliness*, which compares the speed of a *single* process p to the speed of another process q, as follows: p *is timely with respect to* q if, for some integer i, every interval that contains i steps of q contains at least one step of p [3]. Process timeliness, however, cannot be used to study problems that are weaker than consensus such as *set agreement*: the existence of a single process p that is timely with respect to another process q is sufficient to solve consensus in read-write shared memory systems where at most one process may crash (this follows from results in [1, 3]). In fact, all the partially synchronous systems that were previously proposed for message-passing and read-write shared memory are strong enough to solve consensus (under some condition on the number of processes that may crash).

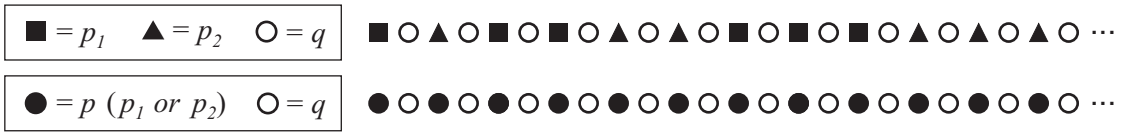

Figure 1: Example of set timeliness. Top shows a schedule with three processes, p_1, p_2, q, in which neither p_1 nor p_2 is timely with respect to q. Bottom shows the same schedule where p_1 and p_2 are considered as a single virtual process p, and p is timely with respect to q.

In this paper, we propose a simple generalization of process timeliness, called *set timeliness*, and show that it can be used to study and compare the partial synchrony requirements of problems that are weaker than consensus. Intuitively, this generalization is obtained by considering a set of processes P in the system as a *single entity*, i.e., as a "virtual process" p that takes a step whenever any process in P takes a step, and then use the definition of process timeliness on such virtual processes. So, a *set* of processes P is timely with respect to another *set* of processes Q if, for some integer i, every interval that contains i steps of processes in Q contains at least one step of some process in P. As we will see below, the processes in P may not be *individually* timely (i.e., the speed of each process in P may fluctuate beyond any bound), but when they are viewed as a single (cooperating) process they may be timely. So a set of processes may be able to overcome the speed fluctuations of individual members of the set, by working together as a timely virtual process.

A simple example, depicted in Figure 1, illustrates the definition of set timeliness. Consider the synchrony of processes p_1 and p_2 with respect to process q in schedule $S = [(p_1 \cdot q)^i \cdot (p_2 \cdot q)^i]_{i=1}^{\infty}$. Note that p_1 is *not* timely with respect to q in S, because there are longer and longer sequences of consecutive steps in S where q takes more and more steps while p_1 takes no step at all: intuitively, there are longer and longer periods where p_1 is very slow with respect to q. Similarly, p_2 is *not* timely with respect to q in S. But if we consider p_1 and p_2 as a single virtual process p, then the above schedule S now becomes $(p \cdot q)^{\infty}$, and the virtual process p is indeed timely with respect to q. In other words, if p_1 and p_2 are considered as a single entity (a set of two cooperating processes), then together they are timely with respect to q. In our model of partial synchrony, we say that the set of processes $\{p_1, p_2\}$ is timely with respect to the set $\{q\}$. Similarly, a set of processes $\{p_1, p_2\}$ is timely with respect to a set $\{q_1, q_2, q_3\}$ if, when we remove all the indices from these processes, the resulting virtual process p is timely with respect to virtual process q.

In this paper, we show that set timeliness can be used to study the synchrony requirements of sub-consensus tasks. In particular, we use it to define a family of partially synchronous systems, and prove tight possibility/impossibility results for solving the *t-resilient k-set agreement problem* — a well-known generalization of the wait-free consensus problem [10] — in these systems.[1]

More precisely:

1. We define a family of partially synchronous systems, denoted $\mathcal{S}^i_{j,n}$, as follows: $\mathcal{S}^i_{j,n}$ is a read/write shared memory system of n processes where at least one set of processes of size i is timely with respect to a set of processes of size j. The family of partially synchronous systems consists of all $\mathcal{S}^i_{j,n}$ where each of i and j ranges from 1 to n.

2. We solve the following general question: *For any t, k, n and any i and j, is the (t, k, n)-agreement*

[1] Intuitively, with the t-resilient k-set agreement problem for n processes, henceforth denoted (t, k, n)-*agreement*, there are n processes that propose values, and if at most t of them crash, then each non-faulty process must decide on a proposed value such that there are at most k different decision values. The problem parameter t ranges from 1 (which corresponds to tolerating a single failure) to $n - 1$ (which corresponds to wait-freedom), and parameter k ranges from 1 (which corresponds to consensus) to $n - 1$ (which corresponds to set-consensus).

problem solvable in partially synchronous system $\mathcal{S}^i_{j,n}$? The answer to this question is surprisingly simple: (t, k, n)-agreement is solvable in $\mathcal{S}^i_{j,n}$ *if and only if* $i \leq k$ and $j - i \geq (t + 1) - k$.

The above result gives the first partially synchronous system that separates the (t, k, n)-agreement problem from the following two incrementally stronger problems: $(t + 1, k, n)$-agreement, which has a slightly stronger resiliency requirement, and $(t, k - 1, n)$-agreement, which has a slightly stronger agreement requirement. In fact, the result implies that partially synchronous system $\mathcal{S}^k_{t+1,n}$ is synchronous enough for solving (t, k, n)-agreement, but not enough for solving $(t + 1, k, n)$-agreement or $(t, k - 1, n)$-agreement. The partially synchronous systems that "closely match" the $(t + 1, k, n)$-agreement and $(t, k - 1, n)$-agreement problems are $\mathcal{S}^k_{t+2,n}$ and $\mathcal{S}^{k-1}_{t+1,n}$, respectively.

Our work is related to results in the IIS and IRIS models [5, 18, 19]. We discuss this and other related work in Section 6.

Roadmap. This paper is organized as follows. In Section 2, we define the notion of set timeliness and use it to define the partially synchronous system $\mathcal{S}^i_{j,n}$. In Section 3, we describe the (t, k, n)-agreement problem. In Section 4, we prove that (t, k, n)-agreement is solvable in system $\mathcal{S}^k_{t+1,n}$. In Section 5, we determine when (t, k, n)-agreement is solvable in system $\mathcal{S}^i_{j,n}$. In Section 6, we discuss related work.

2 Model

We consider a shared-memory system with n processes $\Pi_n = \{1, \ldots, n\}$, which can communicate with each other via some (possibly infinite) set Ξ of shared registers.

A *schedule* S (in Π_n) is a finite or infinite sequence of processes (in Π_n). A *step of a schedule* is an element of S. Given a finite schedule S and a schedule S', we denote by $S \cdot S'$ the concatenation of S and S'. Given an infinite schedule S, a process p is *correct in* S if there are infinitely many occurrences of p in S, and p is *faulty in* S otherwise (in this case, we also say that p *crashes in* S).

2.1 Set timeliness

In what follows, P, P', Q, and Q' are sets of processes in Π_n and S is a schedule in Π_n.

Definition 1 *P is timely with respect to Q in S if there is an integer i such that every sequence of consecutive steps of S that contains i occurrences of processes in Q contains a process in P.*

The following observations follow directly from the above definition:

Observation 2 *If P is timely with respect to Q in S, and P' is timely with respect to Q' in S, then $P \cup P'$ is timely with respect to $Q \cup Q'$ in S.*

Observation 3 *If P is timely with respect to Q in S, and $P \subseteq P'$ and $Q' \subseteq Q$, then P' is timely with respect to Q' in S.*

The definition of *set* timeliness given above (Definition 1) is a direct generalization of the definition of *process* timeliness given in [3]. In fact, Definition 1 can be used to define process timeliness: A process p *is timely with respect to a process q in S* if set $\{p\}$ is timely with respect to set $\{q\}$ in S.

2.2 Systems and partially synchronous systems

A system may be defined by some properties, e.g., timeliness properties, of its schedules. So we define a system $\mathcal{S}$ as a tuple $\mathcal{S} = (\Pi_n, \Xi, \mathit{Scheds})$ where *Scheds* is a set of schedules in Π; intuitively, *Scheds* is the set of schedules that are possible in system $\mathcal{S}$.

The *asynchronous* system of n processes, denoted $\mathcal{S}_n$, is the system $(\Pi_n, \Xi, \mathit{Scheds})$ where *Scheds* is the set of *all* the schedules in Π_n. We define the following family of *partially synchronous systems*: for each i and j such that $1 \leq i \leq j \leq n$, $\mathcal{S}^i_{j,n}$ is the system of n processes where *at least one* set of processes of size i is timely with respect to *at least one* set of processes of size j. More precisely, for every i and j such that $1 \leq i \leq j \leq n$, let $\mathit{Scheds}^i_{j,n}$ be the set of all the schedules S in Π_n such that in S at least one set of processes of size i is timely with respect to at least one set of processes of size j. We define $\mathcal{S}^i_{j,n} = (\Pi_n, \Xi, \mathit{Scheds}^i_{j,n})$.

We say that a *system $\mathcal{S}'$ is contained in system $\mathcal{S}$*, and write $\mathcal{S}' \subseteq \mathcal{S}$, if every schedule of $\mathcal{S}'$ is also a schedule of $\mathcal{S}$, i.e., if $\mathcal{S} = (\Pi_n, \Xi, \mathit{Scheds})$ and $\mathcal{S}' = (\Pi_n, \Xi, \mathit{Scheds}')$ and $\mathit{Scheds}' \subseteq \mathit{Scheds}$.

Observation 3 implies the following:

Observation 4 *For all i, j, n such that $1 \leq i \leq j \leq n$, and all i' and j' such that $1 \leq i' \leq i$ and $j \leq j' \leq n$, $\mathcal{S}^{i'}_{j',n} \subseteq \mathcal{S}^i_{j,n}$.*

Since, in any schedule, every set of i processes is timely with respect to itself, the following is obvious:

Observation 5 *For all i such that $1 \leq i \leq n$, $\mathcal{S}^i_{i,n} = \mathcal{S}_n$, i.e., $\mathcal{S}^i_{i,n}$ is the asynchronous system with n processes.*

2.3 Algorithms and runs

An algorithm $\mathcal{A}$ in a system $\mathcal{S}$ consists of a set of n (infinite or finite) deterministic automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$. By abuse of notation, we identify a process with its automaton. Each process executes by taking steps. In each step, a process p can read or write a shared register and change state (according to p's state transition function in $\mathcal{A}_p$).

Below, $\mathcal{A}$ denotes an algorithm, $\mathcal{S} = (\Pi_n, \Xi, \mathit{Scheds})$ denotes a system, and *pref*(*Scheds*) denotes the set of all finite prefixes of schedules in *Scheds*. A *configuration* of $\mathcal{A}$ in $\mathcal{S}$ indicates the state of each process and register. A *run* R of $\mathcal{A}$ in $\mathcal{S}$ is a tuple $R = (I, S, \mathcal{A})$ where I is an initial configuration of $\mathcal{A}$ in $\mathcal{S}$ and S is a schedule in *Scheds*. A *partial run* P of $\mathcal{A}$ in $\mathcal{S}$ is a tuple $P = (I, S, \mathcal{A})$ where I is an initial configuration of $\mathcal{A}$ in $\mathcal{S}$ and S is a schedule in *pref*(*Scheds*). The *configuration at the end of P* is the state of each process and register after they have taken steps from I in the order indicated by S and according to the state transitions of $\mathcal{A}$. Given a schedule S' where $S \cdot S' \in \mathit{pref}(\mathit{Scheds})$, we denote by $P \cdot S'$ the partial run $(I, S \cdot S', \mathcal{A})$ of $\mathcal{A}$ in $\mathcal{S}$. A *continuation of P in $\mathcal{S}$* is a run $R = (I, S', \mathcal{A})$ of $\mathcal{A}$ in $\mathcal{S}$ where S is a prefix of S'.

3 t-resilient k-set agreement for n processes

Let $1 \leq t \leq n-1$ and $1 \leq k \leq n$. The *t-resilient k-set agreement for n processes* problem, denoted (t, k, n)-*agreement*, is defined as follows. Each process in Π_n has an initial value and must decide a value such that

- *(Uniform k-agreement)* Processes decide at most k distinct values;

- *(Uniform validity)* If some process decides v then v is the initial value of some process; and
- *(Termination)* If at most t processes are faulty then every correct process eventually decides some value.

Note that $(t, n-1, n)$-agreement is also called *t-resilient set agreement*, and $(t, 1, n)$-agreement is also called *t-resilient consensus*. When $t = n-1$, we get the *wait-free* versions of these problems, which are simply called *set agreement* and *consensus*, respectively. In the *binary* versions of all these problems, the initial values of processes are restricted to be in $\{0, 1\}$.

Observation 6 *For all $1 \leq t \leq n-1$ and $1 \leq k \leq n$, if (t, k, n)-agreement can be solved in a system $\mathcal{S}$ then it can also be solved in every system $\mathcal{S}'$ such that $\mathcal{S}' \subseteq \mathcal{S}$.*

Observations 4 and 6 imply the following:

Observation 7 *For all $1 \leq t \leq n-1$ and $1 \leq k \leq n$, if (t, k, n)-agreement can be solved in a system $\mathcal{S}^i_{j,n}$, where $1 \leq i \leq j \leq n$, then it can also be solved in every system $\mathcal{S}^{i'}_{j',n}$ such that $1 \leq i' \leq i$ and $j \leq j' \leq n$.*

4 Solving t-resilient k-set agreement for n processes in system $\mathcal{S}^k_{t+1,n}$

To show that t-resilient k-set agreement for n processes can be solved in $\mathcal{S}^k_{t+1,n}$, we use the t-resilient version of k-anti-Ω — a failure detector given in [21]. In the following, we define t-resilient k-anti-Ω, we give an algorithm that implements t-resilient k-anti-Ω in system $\mathcal{S}^k_{t+1,n}$, and we observe that, from a result in [21], t-resilient k-anti-Ω can be used to solve (t, k, n)-agreement.

4.1 Failure detector k-anti-Ω

Let t and k be such that $1 \leq t \leq n-1$ and $1 \leq k \leq n-1$. With the t-resilient k-anti-Ω failure detector, every process p has a local variable *fdOutput*$_p$ that holds a set of $n-k$ processes, such that the following property holds: if at most t processes are faulty then there exists a correct process c and a time after which, for every correct process p, c is not in *fdOutput*$_p$. Note that when $t = n-1$, t-resilient k-anti-Ω is just the k-anti-Ω failure detector defined in [21].[2]

4.2 Algorithm for t-resilient k-anti-Ω in system $\mathcal{S}^k_{t+1,n}$

We now give an algorithm that implements t-resilient k-anti-Ω in system $\mathcal{S}^k_{t+1,n}$, that is, the algorithm works if every run has at two sets P and Q of sizes k and $t+1$, respectively, such that P is timely with respect to Q. In the following, $\mathbf{\Pi}^k_n$ denotes the set of all subsets of Π_n of size k. The basic idea of our algorithm is that each process p has a heartbeat that it increments periodically, and process p has a timeout timer on each set A in $\mathbf{\Pi}^k_n$. Process p resets the timer for A whenever it sees that the heartbeat of *any* process in A has increased. If p's timer for A expires (the process times out on A), process p increments the timeout that it subsequently uses for A, and p also increments a shared register *Counter*$[A, p]$. This shared register represents a "badness" counter for A as seen by process p. Note that *Counter*$[A, p]$ is monotonically nondecreasing, so either it grows to infinity or it eventually stops changing. We define the accusation counter of a set A to be the $(t+1)$-st smallest value of *Counter*$[A, *]$. Intuitively, the accusation counter of A has two properties: (1) if at least $n-t$ entries of *Counter*$[A, *]$ grow to infinity then the accusation counter of

[2] So $(n-1)$-resilient 1-anti-Ω is equivalent to failure detector Ω [9], and $(n-1)$-resilient $(n-1)$-anti-Ω is also called anti-Ω [21].

A also grows to infinity, and (2) if at least $t+1$ entries of *Counter*$[A, *]$ eventually stops changing then the accusation counter of A also eventually stops changing. Each process p picks the set that has the smallest accusation counter, breaking ties using some arbitrary total order on $\mathbf{\Pi}_n^k$. This set is denoted *winnerset*$_p$, and p outputs the set $\Pi_n -$ *winnerset*$_p$ as the output of k-anti-Ω.

The detailed algorithm is shown in Figure 2. Each process executes an infinite loop, in which the process reads *Counter*$[A, q]$ for each set A in $\mathbf{\Pi}_n^k$ and each process $q \in \Pi_n$, calculates the accusation counter of each set A, chooses a winner, and sets the output of k-anti-Ω accordingly. The process then increments its heartbeat, checks the heartbeats of each process q and, if the heartbeat has increased, it resets the timers of all the sets in $\mathbf{\Pi}_n^k$ containing q. Finally, process p checks if the timers have expired, and increments *Counter*$[A, p]$ for the sets A whose timer expired.

Intuitively, this algorithm works because there is at least one set P of size k that is timely with respect to some set Q of size $t+1$. As we shall see, this implies that eventually every process $q \in Q$ stops increasing *Counter*$[P, q]$. So, at least $t+1$ entries of *Counter*$[P, *]$ eventually stops changing. Thus, the accusation counter of P also eventually stops changing. Among all sets whose accusation counter stops changing, one of them, say A_0, ends up with the smallest accusation counter, and eventually all correct processes pick this set as the winner and output $\Pi_n - A_0$. Note that A_0 must have a correct process: if all processes in A_0 were faulty then all correct processes (there are at least $n-t$ of them) would keep timing out on A_0 and so at least $n-t$ entries of *Counter*$[A_0, *]$ would grow to infinity, so the accusation counter of A_0 would also grow to infinity.

We now sketch a correctness proof. Let k, t, n be such that $1 \leq k \leq t \leq n-1$. Henceforth, we consider an arbitrary run R of the algorithm of Figure 2 in system $\mathcal{S}_{t+1,n}^k$. In the proof, the local variable var of a process p is denoted by *var*$_p$. Let S be the schedule of run R. Henceforth, "steps" refer to steps in S, and a "correct" or "faulty" refers to a correct or faulty process in S, and if we say that a process crashes, we mean it crashes in S.

We must show that if at most t processes crash then there exists a correct process c and a time after which, for every correct process p, c is not in *fdOutput*$_p$. Henceforth, suppose that at most t processes crash. Since R is a run in system $\mathcal{S}_{t+1,n}^k$, we can define the following:

Definition 8 *Let A' and B' be sets of size k and $t+1$, respectively, such that A' is timely with respect to B' in S.*

Lemma 9 *Let $A \in \mathbf{\Pi}_n^k$ and suppose that A is timely with respect to some set $B \subseteq \Pi_n$ in S. Then there exists a constant c such that, for every process $b \in B$, every sequence of consecutive steps of S containing c steps of processes in B contains a step of a process in A that writes in line 7.*

PROOF SKETCH. Each loop interaction has a bounded number of steps, so the result follows from the definition of what it means for set A to be timely with respect to B in S. □

Note that, for any $A \in \mathbf{\Pi}_n^k$, *Counter*$[A, q]$ can only be modified by process q, and only by incrementing it. Thus, *Counter*$[A, q]$ is monotonically nondecreasing and we have the following:

Lemma 10 *For every $A \in \mathbf{\Pi}_n^k$ and every $q \in \Pi_n$, either eventually Counter$[A, q]$ stops changing or it grows monotonically to infinity.*

We now give a sufficient condition for *Counter*$[A, q]$ to eventually stop changing.

SHARED REGISTERS

$\forall p \in \Pi_n$: *Heartbeat*$[p] = 0$
$\forall A \in \mathbf{\Pi}_n^k, \forall q \in \Pi_n$: *Counter*$[A, q] = 0$ { $\mathbf{\Pi}_n^k$ is the set of all subsets of Π_n of size k }

CODE FOR PROCESS p:

Local variables

fdOutput = any set of processes of size $n - k$
winnerset $= \emptyset$
myHb $= 0$
$\forall q \in \Pi_n$: *prevHeartbeat*$[q] = 0$
$\forall A \in \mathbf{\Pi}_n^k$: *timeout*$[A] = 1$
$\forall A \in \mathbf{\Pi}_n^k$: *timer*$[A]$ = *timeout*$[A]$
$\forall A \in \mathbf{\Pi}_n^k$: *accusation*$[A] = 0$
$\forall A \in \mathbf{\Pi}_n^k, \forall q \in \Pi_n$: *cnt*$[A, q] = 0$
hbq $= 0$

Main code

```
1   repeat forever
        { choose FD output }
2       for each ⟨A, q⟩ ∈ Π^k_n × Π_n do cnt[A, q] ← read(Counter[A, q])
3       for each A ∈ Π^k_n do accusation[A] ← (t + 1)-st smallest value of cnt[A, ∗]
4       winnerset ← argmin_{A∈Π^k_n}{(accusation[A], A)}      { break ties using a total order on Π^k_n }
5       fdOutput ← Π_n − winnerset

        { bump heartbeat }
6       myHb ← myHb + 1
7       write(Heartbeat[p], myHb)

        { check other processes' heartbeat }
8       for each q ∈ Π_n do
9           hbq ← read(Heartbeat[q])
10          if hbq > prevHeartbeat[q] then
11              for each A ∈ Π^k_n do
12                  if q ∈ A then timer[A] ← timeout[A]
13              prevHeartbeat[q] ← hbq

        { check for expiration of set timers }
14      for each A ∈ Π^k_n do
15          timer[A] ← timer[A] − 1
16          if timer[A] = 0 then
17              timeout[A] ← timeout[A] + 1
18              timer[A] ← timeout[A]
                { increment Counter[A, p] based on the value read in line 2 }
19              write(Counter[A, p], cnt[A, p] + 1)
```

Figure 2: Algorithm for t-resilient k-anti-Ω in system $\mathcal{S}_{t+1,n}^k$.

Lemma 11 *For every $A \in \mathbf{\Pi}_n^k$ and every $B \subseteq \Pi_n$, if A is timely with respect to B in S then for every process $b \in B$, there is a time after which Counter$[A, b]$ stops changing.*

PROOF SKETCH. From Lemma 9, there exists a constant c such that, every sequence of consecutive steps of S containing c steps of processes in B contains a step of process in A that writes in line 7. In this line, *Heartbeat*$[a]$ is incremented for some $a \in A$. Therefore, for every process $b \in B$, there exists a constant c' such that *timer*$_b[A]$ is reset to *timeout*$_b[A]$ at least once every c' steps of b. Thus, since b increases *timeout*$_b[A]$ each time it finds that *timer*$_b[A] = 0$, there is a time after which b does not find that *timer*$_b[A] = 0$ in line 16. So there is a time after which *Counter*$[A, b]$ stops changing. □

We now give a sufficient condition for *Counter*$[A, q]$ to grow to infinity.

Lemma 12 *For every $A \in \mathbf{\Pi}_n^k$, if every process in A crashes then for every correct process b, Counter$[A, b]$ grows to infinity.*

PROOF SKETCH. If every process in A crashes then eventually no process in A increments its entry in the *Heartbeat* vector. Thus, for every correct process b, there is a time after which b does not set *timer*$_b[A]$ to *timeout*$_b[A]$ in line 12. Then b finds that *timer*$_b[A] = 0$ in line 16 infinitely often, and writes *Counter*$[A, b]$ infinitely often in line 19. Therefore *Counter*$[A, b]$ grows to infinity. □

We now define a pseudo-variable *counter*(A) that depends on the current values of *Counter*$[A, *]$.

Definition 13 *For every $A \in \mathbf{\Pi}_n^k$, counter(A) is the $(t + 1)$-st smallest entry of Counter$[A, *]$.*

Note that since each entry of *Counter*$[A, *]$ is monotonically nondecreasing, *counter*(A) is also monotonically nondecreasing. Thus, we can define the following:

Definition 14 *For every $A \in \mathbf{\Pi}_n^k$, we define $c(A)$ as follows. If counter(A) grows to infinity then $c(A) = \infty$. Otherwise, counter(A) eventually stops changing and we let $c(A)$ be its final value.*

We now establish a relation between $c(A)$ and the entries of *Counter*$[A, *]$.

Lemma 15 *For every $A \in \mathbf{\Pi}_n^k$, $c(A) = \infty$ if and only if at least $n - t$ entries of Counter$[A, *]$ grow to infinity.*

PROOF SKETCH. Let $A \in \mathbf{\Pi}_n^k$. To show the "if" part of the lemma, suppose that at least $n - t$ entries of *Counter*$[A, *]$ grow to infinity. Then the smallest $t + 1$ entries of *Counter*$[A, *]$ includes at least one entry that grows to infinity. Thus, *counter*(A) also grows to infinity, so $c(A) = \infty$.

We now show the "only if" part of the lemma, by showing its contrapositive. Suppose that fewer than $n - t$ entries of *Counter*$[A, *]$ grow to infinity. Then at least $t + 1$ entries of *Counter*$[A, *]$ eventually stops changing. Thus, eventually the smallest $t+1$ entries of *Counter*$[A, *]$ all stop changing (since an entry either stops changing or it grows monotonically to infinity). Thus, *counter*(A) also eventually stops changing, so $c(A) < \infty$. □

Lemma 16 *For every $A \in \mathbf{\Pi}_n^k$, if A is timely with respect to some set B of size $t + 1$ in S then $c(A) < \infty$.*

PROOF SKETCH. By Lemmas 11 and 15. □

Lemma 17 *For every $A \in \mathbf{\Pi}_n^k$, if every process in A crashes then $c(A) = \infty$.*

PROOF SKETCH. By Lemmas 12 and 15, and the fact that there is at least $n - t$ correct processes. □

We now define A_0 to be the set of k processes with smallest $c(A)$, breaking ties using a total order on $\mathbf{\Pi}_n^k$.

Definition 18 *Let $A_0 = \mathit{argmin}_{A \in \mathbf{\Pi}_n^k}\{(c(A), A)\}$.*

Lemma 19 $c(A_0) < \infty$.

PROOF. Recall that set A' is timely with respect to set B' in S, where B' has size $t + 1$. By Lemma 16, $c(A') < \infty$. The result follows since $c(A_0) \leq c(A')$ by definition of A_0. □

Lemma 20 *A_0 has a correct process.*

PROOF SKETCH. Immediate from Lemmas 19 and 17. □

We now establish a relation between $c(A)$ and the local variable $\mathit{accusation}_q[A]$ of a correct process q.

Lemma 21 *For every $A \in \mathbf{\Pi}_n^k$ and every correct process q, if $c(A) < \infty$ then there is a time after which $\mathit{accusation}_q[A] = c(A)$; if $c(A) = \infty$ then $\mathit{accusation}_q[A]$ grows to infinity.*

PROOF SKETCH. Let $A \in \mathbf{\Pi}_n^k$ and q be a correct process. Since q is correct, for every process p, q sets $\mathit{cnt}_q[A, p]$ to $\mathit{Counter}[A, p]$ in line 2 infinitely often. Thus, for each process p, $\mathit{cnt}_q[A, p]$ eventually stops changing if and only if $\mathit{Counter}[A, p]$ eventually stops changing. Thus, by the way q sets $\mathit{accusation}_q[A]$ in line 3 and by definition of $\mathit{counter}(A)$, we have that $\mathit{accusation}_q[A]$ eventually stops changing if and only if $\mathit{counter}(A)$ eventually stops changing. Moreover, if $\mathit{counter}(A)$ eventually stops changing then its final value $c(A)$ is also the final value of $\mathit{accusation}_q[A]$. The result now follows by the definition of $c(A)$: if $c(A) < \infty$ then $\mathit{counter}(A)$ eventually stops changing and so, by the above, $\mathit{accusation}_q[A]$ also stops changing and their final values are the same. if $c(A) = \infty$ then $\mathit{counter}(A)$ grows to infinity, and so $\mathit{accusation}_q[A]$ also grows to infinity. □

Finally, we show that every correct process outputs $\Pi_n - A_0$.

Lemma 22 *There is a time after which every correct process outputs $\Pi_n - A_0$.*

PROOF SKETCH. Let p be any correct process. By Lemma 19, $c(A_0) < \infty$. Thus, by Lemma 21, there is a time after which $\mathit{accusation}_p[A_0] = c(A_0)$.

It is clear that there is a time after which p can only pick A_0 in line 4, because if $A \neq A_0$ then either (a) $c(A) = \infty$, so by Lemma 21 $\mathit{accusation}_p[A]$ grows to infinity, and so there is a time after which $(\mathit{accusation}_p[A], A) > (\mathit{accusation}_p[A_0], A_0)$, or (b) $c(A) < \infty$, so by Lemma 21 and the definition of A_0, there is a time after which $(\mathit{accusation}_p[A], A) = (c(A), A) > (c(A_0), A_0) = (\mathit{accusation}_p[A_0], A_0)$. □

Theorem 23 *For every k, t, n such that $1 \leq k \leq t \leq n-1$, the algorithm in Figure 2 implements t-resilient k-anti-Ω in system $\mathcal{S}^k_{t+1,n}$.*

PROOF SKETCH. Consider any run of the algorithm in Figure 2 in system $\mathcal{S}^k_{t+1,n}$. Suppose that at most t processes crash. It is clear that the output at each process is a set of $n-k \geq 1$ processes. By Lemma 20, there is a correct process c in A_0. By Lemma 22, there is a time after which every correct process outputs $\Pi_n - A_0$, which does not contain c. Hence all the requirements of t-resilient k-anti-Ω are satisfied. □

4.3 Using t-resilient k-anti-Ω to solve (t, k, n)-agreement

A result in [21] implies that t-resilient k-anti-Ω can be used to solve the (t, k, n)-agreement problem in the asynchronous system $\mathcal{S}_n$. By Theorem 23, t-resilient k-anti-Ω can be implemented in system $\mathcal{S}^k_{t+1,n}$. By combining these two results, we have:

Theorem 24 *For every t, k and n such that $1 \leq k \leq t \leq n-1$, the (t, k, n)-agreement problem can be solved in system $\mathcal{S}^k_{t+1,n}$.*

When $t < k \leq n$ it is trivial to solve (t, k, n)-agreement in the asynchronous system $\mathcal{S}_n$. So we have:

Corollary 25 *For every t, k and n such that $1 \leq t \leq n-1$ and $1 \leq k \leq n$, the (t, k, n)-agreement problem can be solved in system $\mathcal{S}^k_{t+1,n}$.*

5 Determining if (t, k, n)-agreement is solvable in $\mathcal{S}^i_{j,n}$

We now present our main result: for every $1 \leq k \leq t \leq n-1$, and every $1 \leq i \leq j \leq n$, we determine whether the (t, k, n)-agreement problem is solvable or not solvable in the partially synchronous system $\mathcal{S}^i_{j,n}$. To do so we first consider the special case where $t = k$, and prove the following theorem:

Theorem 26 *For every k and n such that $1 \leq k \leq n-1$:*

1. *The (k, k, n)-agreement problem can be solved in system $\mathcal{S}^k_{n,n}$.*
2. *The (k, k, n)-agreement problem cannot be solved in system $\mathcal{S}^{k+1}_{n,n}$.*

PROOF SKETCH. Let k and n be such that $1 \leq k \leq n-1$.

1. By Theorem 24, (k, k, n)-agreement can be solved in $\mathcal{S}^k_{k+1,n}$. Since $k+1 \leq n$, by Observation 7, (k, k, n)-agreement can also be solved in $\mathcal{S}^k_{n,n}$.
2. We consider 2 cases:
   (a) $n = k+1$. By a well-known impossibility result given in [4, 13, 20], the $(k, k, k+1)$-agreement problem cannot be solved in the asynchronous system $\mathcal{S}_{k+1}$. By Observation 5, $\mathcal{S}_{k+1} = \mathcal{S}^{k+1}_{k+1,k+1}$.

(b) $n > k+1$. Suppose, for contradiction, that there is an algorithm $\mathcal{A}$ that solves (k, k, n)-agreement in $\mathcal{S}^{k+1}_{n,n}$. We claim that this implies that $(k, k, k+1)$-agreement can also be solved in the asynchronous system $\mathcal{S}_{k+1}$ — contradicting the impossibility result in [4,13,20]. This claim is shown using a simulation algorithm that is similar to those in [6, 7].

Consider the asynchronous system $\mathcal{S}_{k+1}$. The $k+1$ processes of $\mathcal{S}_{k+1}$ can solve $(k, k, k+1)$-agreement by simulating the execution of the algorithm $\mathcal{A}$ in a system $\mathcal{S}$ of $n > k+1$ processes. In this simulation, every schedule S of $\mathcal{S}_{k+1}$ such that at most k processes crash in S maps to a simulated schedule $S_{\mathcal{A}}$ of $\mathcal{S}$ such that:

i. at most k processes crash in $S_{\mathcal{A}}$, and

ii. Every set of $k+1$ processes is timely with respect to the set of n processes in $S_{\mathcal{A}}$, i.e., $S_{\mathcal{A}} \in \mathcal{S}^{k+1}_{n,n}$.

Property (i) was already guaranteed by the simulation algorithms in [6, 7]. We obtain Property (ii) by a careful scheduling of the n simulated threads of algorithm $\mathcal{A}$ by the $k+1$ processes of $\mathcal{S}_{k+1}$.

Let $Simul_{\mathcal{A}}$ be the algorithm that simulates the execution of $\mathcal{A}$ in system $\mathcal{S}$. Let R be an arbitrary run of $Simul_{\mathcal{A}}$ in system $\mathcal{S}_{k+1}$ and S be the schedule of run R. Let $R_{\mathcal{A}}$ be the corresponding simulated run of $\mathcal{A}$ in system $\mathcal{S}$, and $S_{\mathcal{A}}$ be the schedule of run $R_{\mathcal{A}}$.

Suppose at most k processes crash in run R (i.e., in the schedule S of R). By Property (i), at most k processes crash in the corresponding simulated run $R_{\mathcal{A}}$ (i.e., in the schedule $S_{\mathcal{A}}$ of $R_{\mathcal{A}}$). Furthermore, by Property (ii), $S_{\mathcal{A}}$ is in $\mathcal{S}^{k+1}_{n,n}$. Since the algorithm $\mathcal{A}$ solves (k, k, n)-agreement in $\mathcal{S}^{k+1}_{n,n}$ (by our assumption), the simulated run $R_{\mathcal{A}}$ of $\mathcal{A}$, which has schedule $S_{\mathcal{A}} \in \mathcal{S}^{k+1}_{n,n}$, satisfies the properties of the (k, k, n)-agreement problem, namely: (1) every process that is correct in run $R_{\mathcal{A}}$ (i.e., in schedule $S_{\mathcal{A}}$) eventually decides (note that there at at least $n - k > 2$ such processes), (2) all the decision values are initial values, and (3) there are at most k distinct decision values.

Thus, $k+1$ processes can solve $(k, k, k+1)$-agreement in the asynchronous system $\mathcal{S}_{k+1}$ by (1) executing the algorithm $Simul_{\mathcal{A}}$ that simulates some run $R_{\mathcal{A}}$ of the algorithm $\mathcal{A}$ by n processes in system $\mathcal{S}^{k+1}_{n,n}$, and (2) adopting any decision value reached by any of the n processes in this simulated run $R_{\mathcal{A}}$. But solving $(k, k, k+1)$-agreement in $\mathcal{S}_{k+1}$ contradicts the impossibility result in [4, 13, 20].

□

We now state and prove the main result:

Theorem 27 *For every t, k and n such that $1 \leq k \leq t \leq n-1$ and every i and j such that $1 \leq i \leq j \leq n$, the (t, k, n)-agreement problem can be solved in system $\mathcal{S}^{i}_{j,n}$ if and only if $i \leq k$ and $j - i \geq t + 1 - k$.*

PROOF. Let $1 \leq k \leq t \leq n-1$ and $1 \leq i \leq j \leq n$.

1. Suppose $i \leq k$ and $j - i \geq t + 1 - k$. *We show that (t, k, n)-agreement can be solved in $\mathcal{S}^{i}_{j,n}$.*

   We consider 2 cases:

   (a) $j \geq t+1$. By Theorem 24, (t, k, n)-agreement can be solved in system $\mathcal{S}^{k}_{t+1,n}$. Since $i \leq k$ and $j \geq t+1$, by Observation 7, (t, k, n)-agreement can be also solved in system $\mathcal{S}^{i}_{j,n}$.

(b) $j < t+1$. Let S be an arbitrary schedule of system $\mathcal{S}^i_{j,n}$. By definition, in S there is a set of processes P_i of size i that is timely with respect to a set of processes P_j of size j. Since $n \geq t+1$, we have $n-j \geq t+1-j$. So, among the n processes in Π_n, there are at least $t+1-j$ processes that are *not* in the set P_j. Let Q be a set of $t+1-j$ processes that are not in P_j (since $j < t+1$, this set is not empty).

Let $P_{t+1} = P_j \cup Q$ and $P_l = P_i \cup Q$. Since P_j and Q are disjoint, the size of P_{t+1} is $j + (t+1-j) = t+1$. P_i and Q are not necessarily disjoint, so the size of P_l is l such that $i \leq l \leq i + (t+1-j) \leq t+1$. Since P_i is timely with respect to P_j in schedule S, and Q is timely with respect to itself in S, by Observation 2, $P_l = P_i \cup Q$ is timely with respect to $P_{t+1} = P_j \cup Q$ in S. Thus, since $|P_l| = l$ and $|P_{t+1}| = t+1$, every schedule S of $\mathcal{S}^i_{j,n}$ is also a schedule of $\mathcal{S}^l_{t+1,n}$. Hence $\mathcal{S}^i_{j,n} \subseteq \mathcal{S}^l_{t+1,n}$.

By Corollary 25, (t,l,n)-agreement can be solved in $\mathcal{S}^l_{t+1,n}$. Since $\mathcal{S}^i_{j,n} \subseteq \mathcal{S}^l_{t+1,n}$, by Observation 6, (t,l,n)-agreement can also be solved in $\mathcal{S}^i_{j,n}$. By assumption, $j - i \geq t+1-k$, so $k \geq t+1+i-j$ and therefore $k \geq l$. So (t,k,n)-agreement can be solved in $\mathcal{S}^i_{j,n}$.

2. Suppose $i > k$ or $j - i < t+1-k$. *We show that (t,k,n)-agreement cannot be solved in $\mathcal{S}^i_{j,n}$.*

We consider 2 cases:

(a) $i > k$. By Theorem 26 part (2), (k,k,n)-agreement cannot be solved in system $\mathcal{S}^{k+1}_{n,n}$. Since $i \geq k+1$ and $j \leq n$, by Observation 7, (k,k,n)-agreement cannot be solved in $\mathcal{S}^i_{j,n}$. Since $k \leq t$, (t,k,n)-agreement cannot be solved in $\mathcal{S}^i_{j,n}$.

(b) $i \leq k$. Since $i \leq k$, by our hypothesis, we must have $j - i < t+1-k$, and so $1 \leq i \leq k < t+1-(j-i)$.

We claim that (t,k,n)-agreement cannot be solved in $\mathcal{S}^i_{j,n}$. Suppose, for contradiction, that (t,k,n)-agreement can be solved in $\mathcal{S}^i_{j,n}$. We now prove that this implies that, for some $1 \leq \ell < m$, (ℓ,ℓ,m)-agreement can be solved in the asynchronous system $\mathcal{S}_m$ — a contradiction to a well-known impossibility result [6].

Let $\ell = t-(j-i)$ and $m = n-(j-i)$. Since $1 < t+1-(j-i)$ and $n > t$, we have $1 \leq \ell < m$.

Consider the asynchronous system $\mathcal{S}_m$. The $m \geq 2$ processes of this system can solve (ℓ,ℓ,m)-agreement as follows. They pretend they are in a larger system $\mathcal{S}$ with $m + (j-i)$ processes, where the additional $(j-i)$ fictitious processes never take a step. Intuitively, in system $\mathcal{S}$, the $(j-i)$ fictitious processes are crashed from the start. Note that the simulated system $\mathcal{S}$ has a total of $m + (j-i) = n$ processes.

Let P_i be a set of i "real" processes, i.e., they are among the m processes of system $\mathcal{S}_m$,[3] and let C be the set of $(j-i)$ fictitious processes of $\mathcal{S}$.

Now consider any schedule S of the simulated system $\mathcal{S}$. In S it is obvious that the set P_i is timely with respect to itself, and P_i is also timely with respect to the set of crashed processes C. So, by Observation 2, P_i is timely with respect to $P_i \cup C$ in S. Thus, in every schedule S of $\mathcal{S}$, there is a set of size i that is timely with respect to a set of size $i + (j-i) = j$. In other words, every schedule of $\mathcal{S}$ is also a schedule of $\mathcal{S}^i_{j,n}$. So $\mathcal{S} \subseteq \mathcal{S}^i_{j,n}$.

[3]Note that there are at least i processes in $\mathcal{S}_m$, because $m = i + (n-j)$ and $n - j \geq 0$.

By assumption, (t, k, n)-agreement can be solved in $\mathcal{S}^i_{j,n}$. Since $\mathcal{S} \subseteq \mathcal{S}^i_{j,n}$, by Observation 6, (t, k, n)-agreement can also be solved in $\mathcal{S}$. Since $\mathcal{S}$ has $(j - i)$ fictitious processes that are permanently crashed, this implies that $(t - (j - i), k, m)$-agreement can be solved in the "real" system $\mathcal{S}_m$. Since $\ell = t - (j - i)$, (ℓ, k, m)-agreement can be solved in $\mathcal{S}_m$. Since $k \leq t - (j - i) = \ell$, (ℓ, ℓ, m)-agreement can be solved in $\mathcal{S}_m$. Since $\mathcal{S}_m$ is the asynchronous system with $m \geq 2$ processes and $1 \leq \ell < m$, this contradicts an impossibility result in [6]. Thus, (t, k, n)-agreement cannot be solved in $\mathcal{S}^i_{j,n}$.

□

6 Related work

Dwork, Lynch, and Stockmeyer [11] introduce the concept of partial synchrony. They propose message-passing models in which there are eventual or unknown bounds on message transmission times and on relative process speeds. These bounds must hold between every pair of processes. It is shown that consensus can be solved in these models. Subsequent work [1,2,12,14–17] proposed weaker types of partial synchrony (for message-passing systems) with which consensus can still be solved or Ω can be implemented (Ω is the weakest failure detector for consensus [8]). None of these works have considered models in which sub-consensus problems such as (t, k, n)-agreement can be solved, but consensus cannot.

The work in [3] considers a shared-memory model and defines what it means for a single process p to be timely with respect to another process q in any given schedule. The concept of set timeliness introduced in this paper is a direct generalization of this definition, where individual processes p and q are replaced by sets of processes P and Q.

The IIS model [5] is a round-based model in which, in each round, a process atomically writes a value and obtains a snapshot of the values written by other processes in the round. In this model, set agreement and consensus are impossible. Rajsbaum et al. [18, 19] propose a family of models called IRIS that are weaker than the IIS model. This family is parameterized by a property PR_C on the snapshot values that a process can obtain in a round. This property "restricts the asynchrony" of the system, because the fact that a snapshot cannot return certain values means that the execution cannot proceed in certain ways. Specific IRIS models are given in which wait-free k-set agreement is solvable but wait-free $(k-1)$-set agreement is not, thus providing a separation between these problems.

Our model of partial synchrony differs from the IRIS models in two ways. First, we express synchrony behavior directly via timeliness properties of processes, whereas the IRIS models restrict the allowable executions via properties that snapshots must satisfy. Second, our model is based on read-write shared memory, whereas the IRIS model is based on rounds with immediate snapshots. It is possible to implement these rounds in the read-write shared memory model, but it is unclear how the restricted runs of IRIS map to the timeliness properties of the shared memory model. For instance, a process that never appears in the snapshot of other processes may be a process that is actually timely in the shared memory model that implements IRIS: this process may execute at the same speed as other processes but always start a round a few steps later.

The problem of k-set agreement was first defined in [10]. The wait-free and t-resilient versions of this problem were shown to have no solutions in asynchronous systems in [4, 6, 13, 20].

Acknowledgements. The authors are grateful to the anonymous referees for their many helpful comments.

References

[1] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *ACM Symposium on Principles of Distributed Computing*, pages 328–337, July 2004.

[2] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. On implementing Omega in systems with weak reliability and synchrony assumptions. *Distributed Computing*, 21(4):285–314, October 2008.

[3] M. K. Aguilera and S. Toueg. Timeliness-based wait-freedom: a gracefully degrading progress condition. In *ACM Symposium on Principles of Distributed Computing*, pages 305–314, August 2008.

[4] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *ACM symposium on Theory of computing*, pages 91–100, May 1993.

[5] E. Borowsky and E. Gafni. A simple algorithmically reasoned characterization of wait-free computation (extended abstract). In *ACM Symposium on Principles of Distributed Computing*, pages 189–198, August 1997.

[6] E. Borowsky, E. Gafni, N. A. Lynch, and S. Rajsbaum. The BG distributed simulation algorithm. *Distributed Computing*, 14(3):127–146, October 2001.

[7] T. D. Chandra, V. Hadzilacos, P. Jayanti, and S. Toueg. Generalized irreducibility of consensus and the equivalence of t-resilient and wait-free implementations of consensus. *SIAM Journal of Computing*, 34(2):333–357, 2004.

[8] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[9] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[10] S. Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, June 1993.

[11] C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[12] A. Fernández and M. Raynal. From an intermittent rotating star to a leader. Technical Report 1810, IRISA, Université de Rennes, France, August 2006.

[13] M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks (preliminary version). In *ACM Symposium on Theory of Computing*, pages 111–120, May 1993.

[14] M. Hutle, D. Malkhi, U. Schmid, and L. Zhou. Chasing the weakest system model for implementing Ω and Consensus. *IEEE Transactions on Dependable and Secure Computing*. To appear.

[15] E. Jiménez, S. Arévalo, and A. Fernández. Implementing unreliable failure detectors with unknown membership. *Information Processing Letters*, 100(2):60–63, October 2006.

[16] D. Malkhi, F. Oprea, and L. Zhou. Omega meets Paxos: leader election and stability without eventual timely links. In *International Conference on Distributed Computing*, volume 3724 of *LNCS*, pages 199–213. Springer Verlag, September 2005.

[17] A. Mostefaoui, M. Raynal, and C. Travers. Time-free and timer-based assumptions can be combined to obtain eventual leadership. *IEEE Transactions on Parallel and Distributed Systems*, 17(7):656–666, July 2006.

[18] S. Rajsbaum, M. Raynal, and C. Travers. Failure detectors as schedulers (an algorithmically-reasoned characterization). Technical Report 1838, IRISA, Université de Rennes, France, March 2007.

[19] S. Rajsbaum, M. Raynal, and C. Travers. The iterated restricted immediate snapshot model. In *International Computing and Combinatorics Conference*, volume 5092 of *LNCS*, pages 487–497. Springer, June 2008.

[20] M. E. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM Journal of Computing*, 29(5):1449–1483, 2000.

[21] P. Zielinski. Anti-Omega: the weakest failure detector for set agreement. In *ACM Symposium on Principles of Distributed Computing*, pages 55–64, August 2008.